

Saimaa University of Applied Sciences
Technology Lappeenranta
Degree programme in Information Technology
ICT-entrepreneurship

Toni Pulkkinen

Converting Pet Shows game to Unity environment

Thesis 2015

Tiivistelmä

Toni Pulkkinen

Converting Pet Shows game to Unity environment, 25 sivua

Saimaan ammattikorkeakoulu

Tekniikka Lappeenranta

Tietotekniikka

ICT-yrittäjyys

Opinnäytetyö 2015

Ohjaajat: TkT, lehtori Pasi Juvonen, Saimaan ammattikorkeakoulu, toimitusjohtaja Jani Tietäväinen, Seepia Games Oy

Opinnäytetyössä käännettiin yksi HTML5-pohjaisen Pet Shows pelin minipeleistä Unity-ympäristössä toimivaksi. Tavoitteena oli saada Pet Shows sellaiseen ympäristöön, jossa sitä voisi lähteä kehittämään Android käyttöjärjestelmälle. Samalla myös tutkittiin Unityn Android kehityksen mahdollisuuksia ja vaatimuksia. Opinnäytetyön asiakkaana oli Seepia Games.

Minipeli toteutettiin Unity pelimoottorilla hyödyntäen sen 2D-ominaisuuksia. Ohjelmointikielenä toimi C#. Seepia Games toimitti valmiit kuvat ja spritet, joita käytettiin alkuperäisessä pelissä.

Lopputuloksena saatiin aikaan toimiva minipeli, joka muistuttaa alkuperäisversiota sekä lyhyt ohjeistus siihen, miten päästä alkuun Unityssä, jos halutaan kehittää pelejä Android-alustalle.

Asiasanat: Android, C#, HTML5, minipeli, Unity

Abstract

Toni Pulkkinen

Converting Pet Shows game to Unity environment, 25 pages

Saimaa University of Applied Sciences

Technology Lappeenranta

Degree Programme in Information Technology

ICT-entrepreneurship

Bachelor's Thesis 2015

Instructor(s): D.Sc.(Eng.) Pasi Juvonen, Senior Lecturer, Saimaa University of Applied Sciences, Mr Jani Tietäväinen, CEO, Seepia Games Oy

The purpose of this thesis was to convert one of the minigames in Pet Shows from HTML5-based to work in Unity environment. The goal was to get Pet Shows in an environment where it would be possible to try developing it further for Android operating system. The possibilities and requirements to start developing on Android platform were also researched. The customer of this thesis was Seepia Games.

The minigame was made by using Unity game engine utilizing its 2D-features. The programming was done with C#. Seepia Games provided all of the images and sprites from the original game.

The final result of this thesis was a working minigame that resembled the original version and a small tutorial on how to get started with Android development in Unity.

Keywords: Android, C#, HTML5, minigame, Unity

Contents

1	Introduction	7
2	Used Tools.....	7
2.1	Unity.....	7
2.2	C#	9
2.3	Git	10
3	Pet Shows.....	11
3.1	Main idea of the game	11
3.1.1	Minigame.....	12
3.2	Enemies.....	13
3.3	Programming	14
3.3.1	Spawning enemies.....	14
3.3.2	Enemy movement	16
3.3.3	Hit detection	17
3.3.4	Score.....	19
4	Android in Unity	20
5	Final thoughts	22
	Figures.....	24
	References.....	25

Acronyms abbreviations and notations

2D	Space with two dimensions: length and width
3D	Space with three dimensions that can be combination of three terms of the following: length, width, height, depth and breadth
Android	Mobile operating system
Animation	Series of images that create an illusion of movement
Axis	Lines used in Cartesian coordinate system
Bitbucket	Web-based hosting service for Git or Mercurial projects
C#	Programming language developed by Microsoft
Collider	Component in Unity that is used to define the physical shape of an object and to trigger events
Cross-platform	Attribute for software that can operate on multiple platforms
Function	Modules of code that perform a certain task
Git	Version control system developed by Linus Torvalds
HTML5	Markup language used for structuring and presenting content for the World Wide Web
JavaScript	Programming language developed by Netscape Communications Corporation
Minigame	A short video game often contained within another video game
Physics engine	Software that provides simulation of things such as gravity and collision of objects
Raycast	Used in Unity scripting to send rays that can hit colliders to determine objects location
Rigidbody	Component that can be given to objects in Unity, allowing them to interact with physics and other objects
Script	Small program written for a scripting language or command interpreter
SDK	Software development kit, a set of tools for software development
Sprite	Two-dimensional image or animation

Unity	Cross-platform game engine developed by Unity Technologies
UnityGUI	Feature in Unity that allows you to draw a user interface in game by using scripts
UnityScript	Nickname given by Unity community to JavaScript used in Unity
User interface	Space where the user interacts with the program
XML	Markup language that is both human readable and machine readable

1 Introduction

The aim of this thesis is to convert one of the minigames from a game called Pet Shows to work in Unity environment. In addition, the requirements needed for developing games on Android platform were also researched.

The need for this thesis arose when a local game development company was visited by me and two of my classmates in hopes of possible bachelor's thesis subjects. Seepia Games had developed a game called Pet Shows using Construct 2, which is a HTML5-based game engine. The problem was that they wanted to publish their game to mobile platforms, but at the moment the game had some performance issues on mobile devices. They requested us to convert Pet Shows to work in Unity environment, which would allow them to not only publish their game for Android but other platforms as well and maybe help with performance. Since Pet Shows has a lot of content we agreed that for this thesis one minigame would be enough and we could continue working on the game afterwards if we wanted to. The work was divided into three different subjects so we all could get a thesis subject. It was decided that the subject of this thesis was to reproduce the games mechanics in Unity.

There were four phases in this thesis. The first was to explore the original game and get familiar with the game mechanics. The next phase was to figure out the best way to reproduce all of this in Unity. The third phase included the actual programming. Finally, the possibilities of Android development in Unity environment were researched.

2 Used Tools

This chapter discusses the different tools used during this project. Their background and basic information is provided and why they were chosen for this project explained.

2.1 Unity

Unity is a cross-platform game engine, which means that it is compatible with multiple platforms and devices. It is used to develop video games for PC, con-

soles, mobile devices and websites. It was developed by Unity Technologies in 2005 and has received steady updates since then. At the moment Unity supports 21 different platforms, such as Windows, OS X, Linux, Android, iOS, Windows Phone, PlayStation, Xbox and Wii. There are two different editions of Unity: Personal and Professional. Personal is free, but lacks some of the features that Professional offers. However, if one's annual gross revenue from your games exceeds \$100,000, you have to get the Professional Edition. It costs \$1500 or \$75/month. (Unity (game engine) 2015)

Unity offers multiple features such as physics engine, graphics engine and animation tools. Unity uses Mono for the game engines scripting, and programmers can use JavaScript, C# or Boo for programming. (Unity 3D 2015)

The user interface in Unity (Figure 1) offers a lot of useful information. There are five different views: hierarchy, inspector, project, console and the game itself. Hierarchy shows all the objects currently in your scene. This makes selecting and managing objects easy. New objects can also be added from here. Inspector tells information about the object that is selected. It shows basic information such as the objects location, but also all of the components attached to it. In project view a list of all the folders and files in your project can be seen. Console view shows all warnings and error messages. It gives information about the errors such as what kind of error it is and where in the code it might be located. In the middle of all of this is also the game screen which allows you to playtest your game. Scene window is for editing and moving objects around. Game window shows you what to game looks like for the player. Animator window allows you to edit your game's animations. (Blackman 2013)

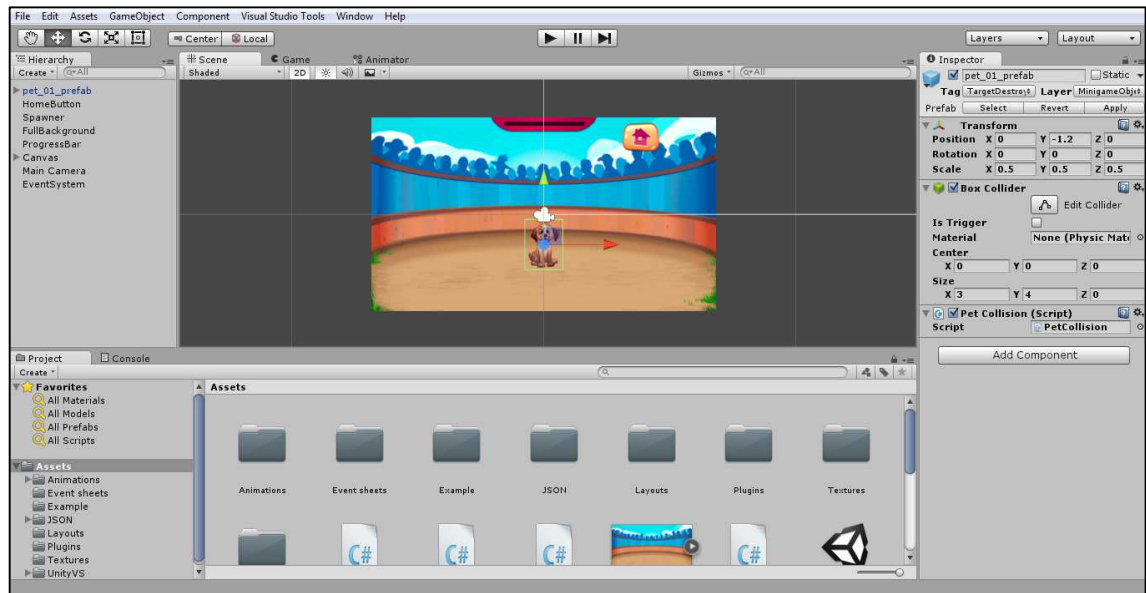


Figure 1 User interface in Unity

Unity was chosen in this project as game engine simply because of its cross-platform features. Not only it helps accomplishing the main goal to make Pet Shows compatible with Android, it also opens possibilities for other platforms as well if Seepia Games would ever wish to pursue them.

2.2 C#

C# is a programming language developed by Microsoft in 2000. The development team is led by Anders Hejlsberg. The goal behind it was to combine the efficiency of C++ and the accessibility of Java. C# is meant to be suited for writing applications for both hosted and embedded systems, from small having dedicated functions to large using sophisticated operating systems. The most recent version C# 6.0 was released on July 20, 2015. (C Sharp (programming language) 2015)

When choosing which programming language to use in this project, it came down to C# or JavaScript since the team had the most experience in these languages. After some research it was decided that C# will be used. The reason is that in Unity programming with JavaScript is a little different. Even the Unity community has started to call it UnityScript, since it is so different compared to JavaScript, that it deserved its own name. To avoid later issues that could arise from not knowing the compatibility between UnityScript and Android, choosing

C# was most likely the right choice. In addition using C# was also recommended by Seepia Games.

2.3 Git

Git is a version control system developed by Linus Torvalds in 2005. It has become one of the most popular version control systems for software development. It supports non-linear development, which means the possibility to create multiple branches where people can work on different things and merge them later to bring all the work together very easily. You can also jump between different versions if something goes wrong. (Git 2015)

During the project Bitbucket (Figure 2) was used to store our project. It is a web-based hosting service for Git or Mercurial projects. Bitbucket was originally an independent startup founded by Jesper Nohr. On 29 September 2010 it was acquired by Atlassian. With Bitbucket, one can create a free account that can have unlimited number of private repositories with five users. Git and Bitbucket were chosen because the team was familiar with them from past projects. (Bitbucket 2015)

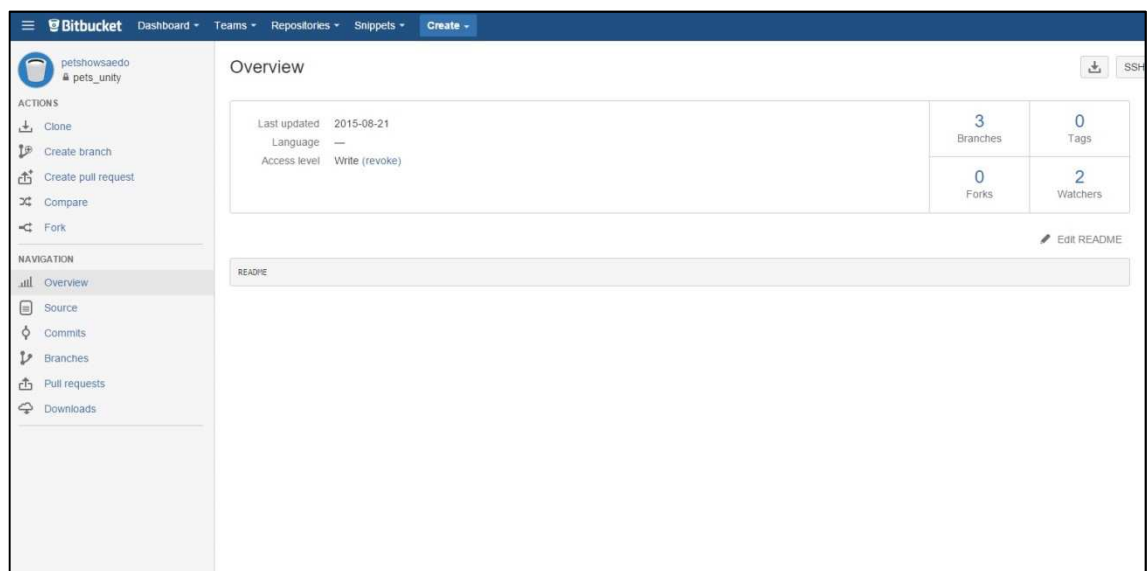


Figure 2 Bitbucket overview

3 Pet Shows

In this chapter it is explained what kind of game Pet Shows is, what the objective is and what kind of content it has. Some of the programming behind the game is also explained.

3.1 Main idea of the game

In Pet Shows the player's goal is to train different kinds of pets and then compete with them in a variety of competitions. Your pets gain experience depending on how much you train, compete and how well you perform in these activities. By training one's pets, the player will also be able to increase their various stats. Doing well in competitions will earn the player different currencies. They can be used to buy items that boost the abilities of the player's pet or they can hire coaches to help in training.

The player can even compete against other players, since all of the results are saved in the database. The game will choose opponents for the player depending on their skill and the player has to beat their scores to succeed in competitions. Players will travel to tournaments to show off their pet's abilities. The tournaments are made of 3 competitions that are selected randomly. Each competition requires 3 pets and the player has to be able to fill all spots to participate. The player can train more pets if they don't have enough. The players will compete in seasons and collect points to advance in scoreboards. The players who place high will receive additional rewards.

After competing many times the, player's pets will grow old and they have to retire, which will lead to training new pets. The game has no clear ending: the player will continue to attempt to succeed and make each pet the best they can be. There is a lot of unlockable content in the game. For example all the pets have rare color variants that might show up when training new pets. From the main menu (Figure 3) the player can get a lot of information, such as your currencies and level, and gain access to all parts of the game.

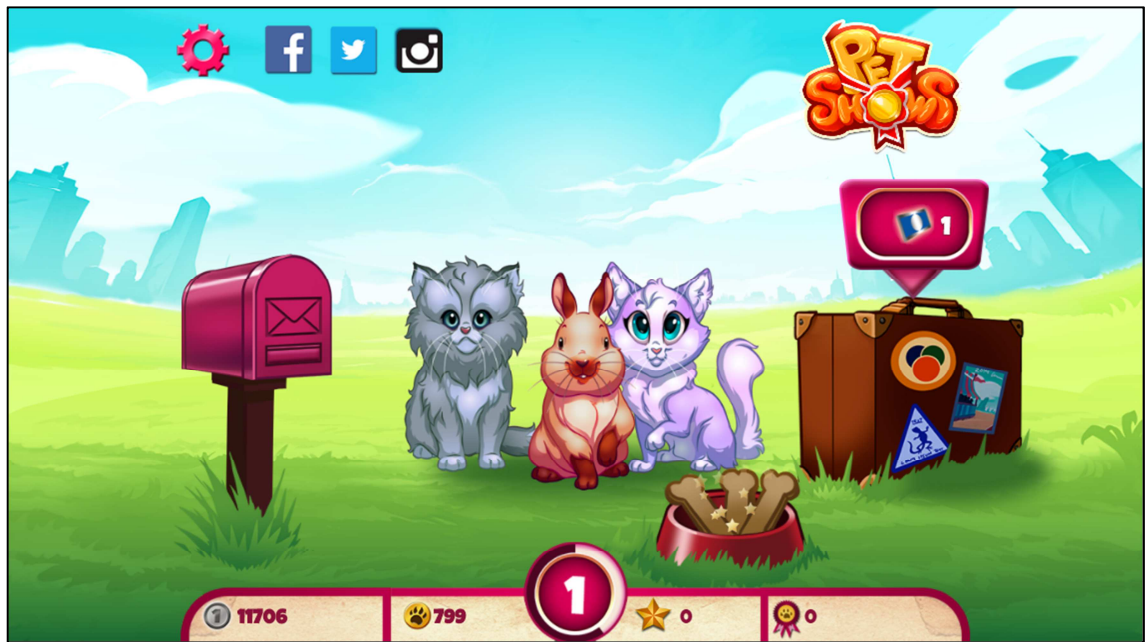


Figure 3 Pet Shows main menu

3.1.1 Minigame

The Pet Shows gameplay is divided into various minigames. In this introduction only the minigame that was chosen for this thesis will be discussed, since the subject of the thesis was to convert only one of them.

The competition of cuteness was chosen as the minigame in this project (Figure 4). In this competition the player's goal is to protect their pets from different kinds of enemies that try to reach them. Enemies consist of different bugs, and to keep them away the player has to hit them with a fly swatter. Bugs will spawn from the edges of the screen and move slowly towards the player's pets. If the player manages to keep their pets safe, the player will slowly accumulate points. However if the enemies reach the player's pets they will lose points. If one of the player's pets takes too many hits, it is removed from the arena.

The number and types of enemies depend on the difficulty level. More difficulty levels will be unlocked by succeeding in competitions. The minigame will end after there are no more enemies left, or all of the pets are removed from the arena. If the player manages to keep their points over the set point, they will win.



Figure 4 Minigame

3.2 Enemies

There are three types of enemies in the minigame: small flies, big flies and ground enemies. Each type also has two kinds of variants. To simplify this, these enemies will be separated into two categories based on their behavior. These categories are flying enemies and ground enemies.

Flying enemies (Figure 5) consist of smaller flies, which are the most common enemy, and bigger flies. Their difference between these enemies is their size. Their behavior is the same and they both can be defeated with just one swatter hit. Bigger flies are only another type of enemy to increase the number of enemies on later difficulties. Flying enemies move in a swaying motion to make them harder to hit.



Figure 5 Flying enemies

There are only two kinds of ground enemies (Figure 6). They are the easiest to defeat, but make the player lose more points compared to flying enemies if they reach the pets. They are slower than flying enemies and move in a straight line, unlike their flying counterparts. They are not more durable than flying enemies even though their appearance would suggest otherwise. Only one swatter hit is enough to defeat them.



Figure 6 Ground enemies

3.3 Programming

C# was used to program the game in Unity. This chapter will showcase the most essential parts of the code.

3.3.1 Spawning enemies

In the original game enemies will spawn from edges of the screen in set intervals. The location where the enemy spawns doesn't follow any logical pattern so it had to be made completely random. In addition there seemed to be a set amount of enemies that spawn for different difficulties. Three different difficulties were made for testing purposes and also for future use, if the conversion of the game was continued.

In the part of the script (Figure 7) that determines the location of enemy spawn, a timer for each enemy is set to control their spawn times. The script will not do anything until the timer hits a certain point of time. After the timer is triggered, random generator is used to determine which edge of the screen the enemy will spawn from. Then random generator is used again to decide the location of the spawn. (Sempf, Sphar & Davis 2010)

```

48 // Update is called once per frame
49 void Update () {
50     // start spawn timers
51     spawn1Timer += Time.deltaTime;
52     spawn2Timer += Time.deltaTime;
53     spawn3Timer += Time.deltaTime;
54
55     // check if enough time has gone by since last enemy spawn and if there is any spawns left
56     if (spawn1Timer > spawn1Cooldown & spawn1Count > 0){
57         // determine which edge of the screen the enemy spawns
58         spawnDirection = Random.Range(0,3);
59         switch (spawnDirection) {
60             // when edge of the screen is selected determine the location of enemy spawn
61             case 0: {
62                 spawnLocationX = -7.5f;
63                 spawnLocationY = Random.Range(-4.5f, 4.0f);
64                 break;
65             }
66             case 1: {
67                 spawnLocationX = 7.5f;
68                 spawnLocationY = Random.Range(-4.5f, 4.0f);
69                 break;
70             }
71             case 2: {
72                 spawnLocationX = Random.Range(-7f, 7f);
73                 spawnLocationY = -4.5f;
74                 break;
75             }
76         }

```

Figure 7 Spawn enemies part 1

Now that the location has been determined, it is time to proceed to the part of the script (Figure 8) where enemies are created. Since each enemy type has two alternative sprites, first it must be decided which one will be used. This is determined by using random generator again. Finally, a copy of the enemy sprite is created at the location determined before. In the end, the timer is reset so the script can spawn the next enemy. In addition, the enemy that spawned is subtracted from the total spawn count. When the spawn count hits zero, no more enemies will spawn and the game ends.

```

77 // select the location
78 transform.position = new Vector2(spawnLocationX, spawnLocationY);
79 // determine which sprite is used
80 spawnAlt = Random.Range(0,2);
81 switch (spawnAlt){
82     // create the enemy depending on which sprite was selected
83     // reset timers and subtract from spawn count
84     case 0: {
85         Instantiate(spawn1, transform.position, transform.rotation);
86         spawn1Timer = 0f;
87         spawn1Count -= 1;
88         break;
89     }
90     case 1: {
91         Instantiate(spawn1Alt, transform.position, transform.rotation);
92         spawn1Timer = 0f;
93         spawn1Count -= 1;
94         break;
95     }
96 }

```

Figure 8 Spawn enemies part 2

3.3.2 Enemy movement

In the minigame, enemies will move from the edge of the screen towards the pets located in the middle. The first thing to do is to determine which side of the screen the enemy spawned and turn it accordingly so it faces the middle. This cannot be done by simply ordering the enemy to look at the middle because 2D sprites are used. What has to be done in this part of the script (Figure 9) is to change the scale value of the sprite from positive to negative. The result is a sprite that has been turned inside out making it to face the opposite direction.

```
10 public float speed = 1f; // Controls the speed of the object
11 private Vector3 target = new Vector3(0f, -1.2f, 0f); // Object will move towards this location
12 private int direction; // Changes the scale of the object depending on which side
13
14 private Vector3 yForce; // Vector that does the swaying motion
15 private float yForceMovement = 1f; // Controls the up and down movement
16 private float yForceDirection = 1f; // Determines if the object is moving up or down
17 private float yDirectionTimer = 0f; // Timer that triggers the change in direction
18
19 // Use this for initialization
20 void Start () {
21     // checks which side of the screen the object is and change its scale accordingly so it turns in the right direction
22     if (transform.position.x < 0){
23         direction = -1;
24     }
25     else if (transform.position.x > 0){
26         direction = 1;
27     }
28     transform.localScale = new Vector3(direction, 1, 1);
29 }
30
```

Figure 9 Enemy movement part 1

For ground enemies the movement part is fairly simple. A target location is determined and the enemy is commanded to move towards it at certain speed. For flying enemies the script (Figure 10) is slightly more complicated. Since flying enemies move in a swaying motion, some additional actions have to be done to make them work as intended. An additional force has to be added to control the up and down swaying. First, a vector is created to make the swaying motion while the enemy moves towards the middle at the same time. By making the vector's y-axis positive, the enemy will move up when it first spawns. Next, the vector's y-axis is saved in a variable. A certain amount is subtracted from this value at set intervals, so with time it will eventually turn negative. This means that the enemy will slowly start moving down. A timer has also been placed that controls when it is the right time to subtract or add to the variable. The timer is set in a way that it will start adding to the variable at exactly the right time making the sway movement very smooth and balanced. (Unity Answers)


```

31 // Update is called once per frame
32 void Update () {
33     // makes the object move at the target location with certain speed
34     float step = speed * Time.deltaTime;
35     transform.position = Vector3.MoveTowards(transform.position, target, step);
36
37     // Controls the force that does the swaying motion
38     yForce.y = yForceMovement;
39     transform.position += yForce * Time.deltaTime;
40     yForceMovement -= yForceDirection * Time.deltaTime;
41     // Timer that controls when the swaying motions direction is changed
42     yDirectionTimer += Time.deltaTime;
43     if (yDirectionTimer > 2f) {
44         yForceDirection = yForceDirection * -1;
45         yDirectionTimer = 0f;
46     }
47 }
48

```

Figure 10 Enemy movement part 2

3.3.3 Hit detection

There are two situations in the game where hit detection is needed: when the enemy is hit by the fly swatter or when the enemy reaches the pets. The programming to detect if something has been hit isn't complex. Most of the work is done by the physics engine in Unity.

In Unity, one can add different components to the objects in one's game. First component that has to be added to our objects for hit detection is rigidbody. This component allows objects to interact with physics, and more importantly in this situation, with other objects. Now that the game's objects can interact with each other, some boundaries have to be set for the objects. This can be done with a component called collider. Colliders are used to define the physical shape of an object. It can also be used to trigger events when other objects come inside its boundaries. This is used in our script (Figure 11) to destroy enemies when they come in contact with the pets or the flyswatter. First, a tag is set in Unity for the pets and the fly swatter to later be able to identify them in the code as triggers for the event that destroys the enemies. The code itself is fairly simple. However, it should be mentioned that the same event is used to spawn our splatter sprite when the enemies get destroyed.

```

49 void OnTriggerEnter (Collider other) {
50     // check objects tag
51     if (other.tag == "TargetDestroyer") {
52         // determine which splatter sprite is used
53         splatterAlt = Random.Range(0,2);
54         if (splatterAlt == 0) {
55             Instantiate(splatter1, transform.position, transform.rotation);
56         }
57         if (splatterAlt == 1) {
58             Instantiate(splatter2, transform.position, transform.rotation);
59         }
60     }
61 }

```

Figure 11 Hit detection part 1

The first script was used to determine when the enemies should get destroyed, however the game must also know when a player clicks with the mouse, and if there is something under the cursor worthy of hitting. Something that has to be ensured as well is that the player cannot use the fly swatter continuously. There has to be some kind of punishment for missing the target. In this script (Figure 12), raycast is used to determine if there is an object under the cursor. Raycast sends a ray from its origin until it hits a collider. In this case, the origin is the cursor and the ray is sent under it. If an object is hit the, fly swatter animation can be started.

```

4 public class HitDetector : MonoBehaviour {
5
6     public Transform swatter;
7
8     private float swatterTimer = 0f; // timer that checks if flyswatter can be used
9     public float swatterCooldown = 1f; // controls how often the flyswatter can be used
10
11     Ray ray;
12     RaycastHit hit;
13
14     // Use this for initialization
15     void Start () {
16
17     }
18
19     // Update is called once per frame
20     void Update () {
21         swatterTimer = swatterTimer + Time.deltaTime;
22         // checks if mouse is clicked
23         if (Input.GetMouseButton(0)) {
24             // checks if the flyswatter is on cooldown
25             if (swatterTimer > swatterCooldown) {
26                 // sends a ray from the mouse position to check if there is a object under the cursor
27                 ray = Camera.main.ScreenPointToRay(Input.mousePosition);
28                 // if object is hit start the flyswatter animation
29                 if (Physics.Raycast(ray, out hit)) {
30                     Instantiate(swatter, hit.point, transform.rotation);
31                     swatterTimer = 0f;
32                 }
33             }
34         }
35     }
36 }

```

Figure 12 Hit detection part 2

3.3.4 Score

Players accumulate score as long as their pets are still in the game and there are still enemies left. Score can be seen as a bar on top of the screen. To draw the score bar a feature in Unity called UnityGUI was used.

UnityGUI allows the programmer to draw a user interface in game by using a script (Figure 13). There are a lot of different ways to use this feature, but in this case it is only needed for drawing the score bar. First, the location and size of the bar is selected. In the OnGUI function the frame of our bar is drawn. By doing this it can then be utilized to determine where the bar itself will be drawn using the size and texture selected.

```
4 public class ProgressBar : MonoBehaviour {
5
6     public Texture2D Tex;
7
8     public static float barDisplay;           // current score
9     private float barSpeed;                  // speed for filling the score bar
10    public Vector2 pos = new Vector2(20,40);   // vector used to determine the location of the bar
11    public Vector2 size = new Vector2(60,20);  // vector used to determine the size of the bar
12
13    void Start () {
14        // determine the position and size of the score bar
15        pos = new Vector2(561,12);
16        size = new Vector2(250,13);
17
18        // set score to 0 and start accumulating score incase of a new game
19        barSpeed = 0.00075f;
20        barDisplay = 0f;
21    }
22
23    void OnGUI() {
24        // draw the frame for our bar
25        GUI.BeginGroup(new Rect(pos.x, pos.y, size.x, size.y));
26        //draw the bar
27        GUI.BeginGroup(new Rect(0,0, size.x * barDisplay, size.y));
28        GUI.DrawTexture(new Rect(0,0, size.x, size.y), Tex);
29        GUI.EndGroup();
30        GUI.EndGroup();
31    }
```

Figure 13 Score part 1

Now that the groundwork has been done, manipulating the bar in the next part of the script (Figure 14) can begin. If the score bar is not full, certain amount is added to it at certain speed. If there are no more enemies left, the speed is set to 0.

```

33 // Update is called once per frame
34 void Update () {
35     // check if the bar is full and accumulate points incase its not
36     if(barDisplay < 0.999f){
37         barDisplay += barSpeed;
38     }
39     // if there is no more enemies left stop accumulating points
40     if(SpawnTarget.spawn1Count == 0 & SpawnTarget.spawn2Count == 0 & SpawnTarget.spawn3Count == 0){
41         barSpeed = 0f;
42     }
43 }

```

Figure 14 Score part 2

4 Android in Unity

If one wants to develop a game for multiple platforms, Unity is one of the best choices. With its cross-platform feature one can finish their game and then build it afterwards for any platform that is compatible with Unity.

When developing a game with mobile platforms in mind, there are a couple of things to consider. The first is how to control the game, since on mobile devices there is limited amount of inputs available. The second is being aware what kind of hardware is needed to run the game and which screen resolution one is aiming for, because those are quite limited on mobile devices, as well.

When everything has been taken into consideration, it is time to set up a mobile development environment. For Unity to match the wanted development environment, the corresponding SDK has to be downloaded. In this case, it would obviously be Androids SDK, which is available for download on their website. Next, naturally before developing can begin, Unity must know where the Android SDK is located before all of its features can be used. This can be done in Unity options (Figure 15) by first selecting Edit and then Preferences. After that, there is a section called External Tools, where the location of Android SDK can be browsed. (Unity Mobile development)

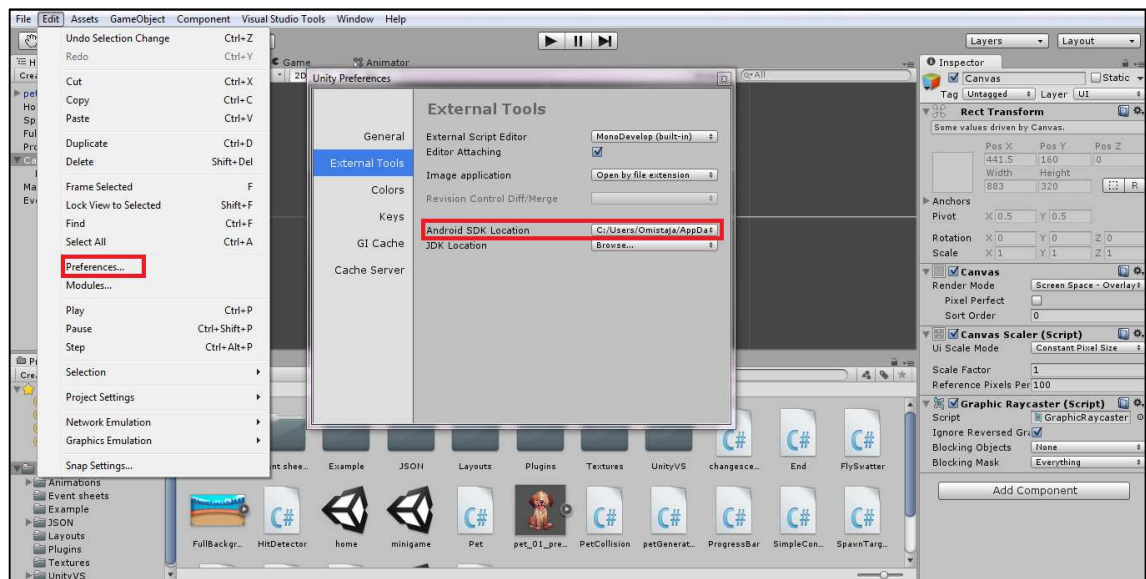


Figure 15 Setting up Android SDK

The next step is to make sure that Unity is set up for the wanted development platform. Clicking File (Figure 16) and then Build Settings brings up a window where the correct platform can be selected from the list. When all is done clicking Switch Platform saves the changes. It is important that all the scenes have been added to the box at the top for being able to change scenes during the game. The actions described are the required steps for getting started with mobile development. With android SDK installed programming of touchscreen features and such is possible.

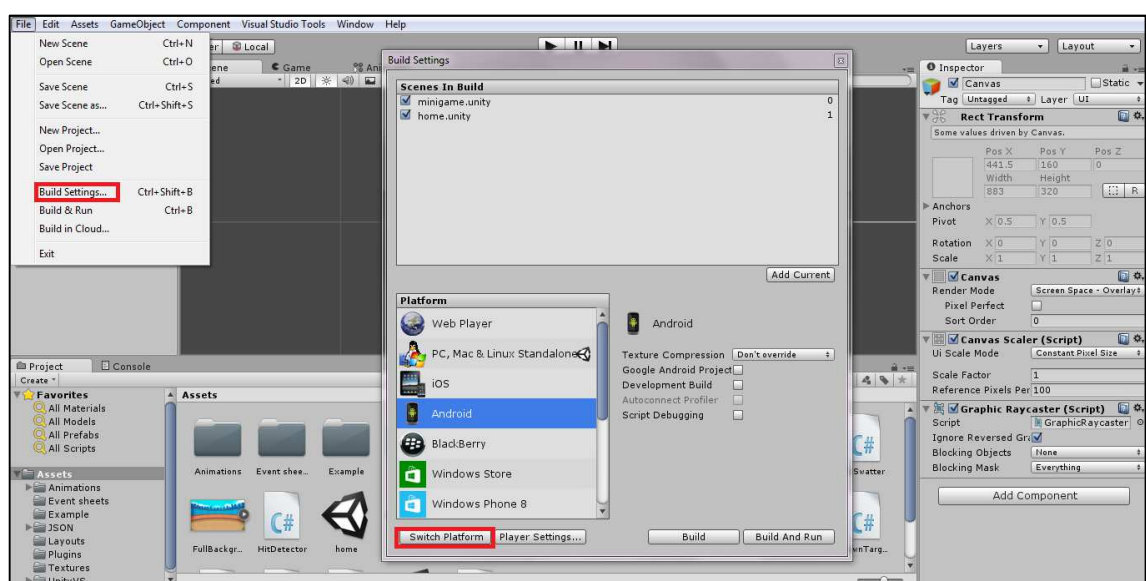


Figure 16 Build Settings

5 Final thoughts

The main goal of this thesis was to convert a minigame from Pet Shows to an environment that works with Android. This was achieved by using Unity as it allows one to build their game for multiple platforms. The final result of the project works a lot like the original game, even though the project had to be basically started from the beginning. The tools selected for this project were all available for free and they also fulfilled other requirements, such as that the members of the team were already familiar with them. This made the process much easier.

The process of this thesis was quite successful. Four phases that were listed at the start of this thesis will be used to discuss it. The first phase of getting to know the game and its mechanics was easy to accomplish, because Pet Shows can be played in a browser. This made studying how the game worked very effortless, since the game could be accessed from anywhere and anytime I needed.

However, there were some problems during the project. The first challenge was in the second phase: studying the original code. The original game data that we got from Seepia Games was in XML form. The team was not familiar with this type of file, so its contents were quite hard to understand. At some points it would have been helpful to know certain things, such as different timers used in the original game. In the end it was decided to start the project entirely from the beginning and solve problems as they surface. After the work was already done the team had a discussion with the representatives of Seepia Games and it was learned that if one opened these files using Construct 2 it would be much easier to understand their contents. Now looking back, it might have facilitated the project. It encouraged exploring different methods, instead of being stuck with set rules. As a result a lot more experience was gained.

The biggest problem in the project was in the third phase with the 2D features in Unity. For some reason when trying to use 2D components for the game objects, they did not interact with each other as they should have. Some research was done and it was found out that other Unity users had similar problems. In

the end it was decided to use 3D components in some parts of the game as a good solution could not be found. This did not really affect anything as everything still works the same way as they would with 2D components. The only thing 2D components would have done is that with them the game would only have to worry about two dimensions instead of three. This would slightly lower the performance requirements, but in a small minigame like this it probably will not make any difference. If the project is continued in the future this might be looked at in further details. The problem might have been fixed in a patch was not used, because it was decided not to update the programs. This prevented any issues that could arise from team members having different versions.

The fourth phase was to research the possibilities of Android development. This was quite easy task to accomplish, since Unity has so many tutorials on its website. With some time, a really good tutorial was found that explained how to get started with mobile development and it also had some programming examples for a touch screen.

Figures

Figure 1. User interface in Unity, page 9

Figure 2. Bitbucket overview, page 10

Figure 3. Pet Shows main menu, page 12

Figure 4. Minigame, page 13

Figure 5. Flying enemies, page 13

Figure 6. Ground enemies, page 14

Figure 7. Spawn enemies part 1, page 15

Figure 8. Spawn enemies part 2, page 15

Figure 9. Enemy movement part 1, page 16

Figure 10. Enemy movement part 2, page 17

Figure 11. Hit detection part 1, page 18

Figure 12. Hit detection part 2, page 18

Figure 13. Score part 1, page 19

Figure 14. Score part 2, page 20

Figure 15. Setting up Android SDK, page 21

Figure 16. Build Settings, page 21

References

Bitbucket. <https://en.wikipedia.org/wiki/Bitbucket>. Accessed on 24 August 2015

Blackman, S. 2013. Beginning 3D Game Development with Unity 4 All-in-one, multi-platform game development. New York: Apress.

C Sharp (programming language).
[https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)). Accessed on 24 August 2015

Git. [https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software)). Accessed on 24 August 2015

Sempf, B., Sphar, C. & Davis, S.R. 2010. C# 2010 All-in-One For Dummies. Indianapolis: Wiley Publishing Inc.

Unity 3D. <https://unity3d.com>. Accessed on 13 August 2015

Unity Answers. <http://answers.unity3d.com>. Accessed on 31 July 2015

Unity (game engine). [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)). Accessed on 13 August 2015

Unity Mobile development.
<https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/mobile-development>. Accessed on 24 August 2015